D4A Magazine

مجلة منتدى دلفي للعرب

Issue 04

العدد رقم : 04



تعرّف على فايربيرد في دقيقتين



البرمجة بالمنحى للكائن OOP خطوة، خطوة

- نسخ الملفات
- تجسيد المكدس
- التوجيه Inline

مقدمة في برمجة الشبكات باستخدامINDY



فهـــرس العدد



افتتاحيـــــــــــــــــــة

بسم الله الرحمن الرحيم

تتقدم إدارة وفريق العمل بمنتدى دلفي للعرب بأطيب التحيات وخالص التهاني للأمة الإسلامية في جميع أقطار العالم عامة ولأسرة وأبناء المنتدى خاصة، بمناسبة حلول السنة الهجرية الجديدة متمنية لهم قبول الأعمال والدوام على الطاعات والعبادات...

بعد تلقينا ببالغ الحزن خبر انسحاب المشرف الفذ الأخ STRELiTZIA الذي كان يشرف على المجلة، متمنين له النجاح والاستمرار قدما في مشواره العلمي، قمنا بجمع ما تيسر من مواضيع مع إضافة أطروحات متميزة محاولة لإتمام هذا العدد الذي نأمل أن يرق لمستوى متتبعيه...

في هذا العدد...

حاولنا جمع أكبر قدر من المواضيع المتنوعة سعيا لتغطية أكبر قدر ممكن من اهتمامات قراء المجلة، وحرصا على تقديم الأفضل فإننا ننتظر آرائكم، انتقاداتكم، أو أية تنبيهات بخصوص الأخطاء المطبعية أو المنهجية، كما يبقى قسم المجلة في انتظاركم لأي سؤال أو اقتراحات أو موضوع بخصوص محتوى العدد المقبل بإذن الله.

الكاتب: إدارة المنتدى

البرمجة بالمنحى للكائن ـ بقلم خالد الشقروني

OOP خطوة، خطوة



أنت تريد أن تتعرف على أساسيات البرمجة بالمنحى للكائن Object Oriented Programming

أنت لديك بعض الخبرة بالبرمجة و تريد أن تتعرف على المنحى للكائن و تفهمه استنادا إلى خبرتك هذه وانطلاقا مما تعرفه من تقنيات برمجية.

أن تريد أن تتلمس بيدك كيف تكون البرمجة بالمنحى للكائن، و ليس مجرد تعريفات ومصطلحات غامضة وشروحات طويلة مملة.

إذا، و بدون مقدمات ، وبدون تمهيد نظري، ***دعنا نبدأ***

الجولة الأولى

ابدأ مشروعا جديدا في دلفي، ضع زرا Button على نموذج الشاشة ثم نقرة مزدوجة double click ، نحن أمام إجرائية مناولة حدث OnClick للزر. من هنا نضع خربشاتنا للتجريب و الاستكشاف .

نعلم جميعا أنه يوجد في البرمجة مفهوم إسمه متغيرات Variables **يمكن أن نحمل عليها قيم، مثل المتغير الذي سنعرفه الآن:**

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FirstName: string;

begin

end;
```

قمنا بتعريف متغير إسمه FirstName **من نوع** string، (***كلمة نوع تسمى في دلفي*** *Type*)، **يمكنك الآن أن تسند أية قيمة نصية لهذا المتغير، كأن تقول مثلا:** `FirstName := 'Ahmad';`

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FirstName: string;

begin
  FirstName := 'Ahmad';
End;
```

ويمكننا التأكد من ذلك بطبع القيمة التي يحملها أو يشير إليها هذا المتغير FirstName **على سطح النموذج:**

```
begin
  FirstName := 'Ahmad';

  Canvas.TextOut(10, 10, FirstName);
End;
```

نقوم أيضا بإضافة المزيد من المتغيرات:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  FirstName: string;
  LastName: string;
  BirthDate: TDateTime;

begin
  FirstName := 'Ahmad';
  LastName := 'Hamza';
  BirthDate := EncodeDate(1980, 3, 15);

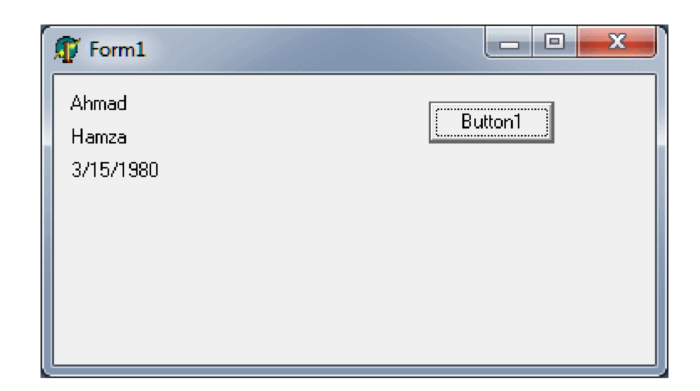
  Canvas.TextOut(10, 10, FirstName);
  Canvas.TextOut(10, 30, LastName);
  Canvas.TextOut(10, 50, DateToStr(BirthDate));

end;
```

كما هو واضح أعلاه بالإضافة إلى FirstName وضعنا متغيرات جديدة: اللقب من نوع string وتاريخ الميلاد من نوع TDate ، ثم خصصنا قيما لهذه المتغيرات، المتغيرات الثلاث معا تمثل حالة فرد ، والقيم تعبر عن فرد بذاته إسمه أحمد حمزة.

ثم كتبنا على شاشة النموذج القيم التي تحملها هذه المتغيرات.

البرنامج بعد التشغيل يكون شكله كالتالي:



الجولة الثانية

المتغيرات التي حددناها سابقا هي من أنواع معرفة مسبقا داخل دلفي مثل string و integer و TDate **وكل نوع له خصائصه، فنوع** string **يحمل سلاسل من أحرف نصية، و** integer **يحمل قيمة بعدد صحيح، و نوع** TDate **يحمل قيمة من نوع تاريخ. وكل متغير نقوم بتحديده سيحمل قيمة توافق النوع الذي انبثق من المتغير. هذا كلام معروف ومفهوم و من أساسيات البرمجة.**

الآن سوف ندفع بهذه المتغيرات إلى مستوى أعلى. ونستخدم نوعا جديدا من أنواع البيانات.

المتغيرات التي عرفناها سابقا؛ سنقوم بضمها في هيكل واحد. هذه الهيكلية تسمى في دلفي record ، **كالتالي:**

```
procedure TForm1.Button1Click(Sender: TObject);

type
  TPerson = record
    FirstName: string;
    LastName: string;
    BirthDate: TDateTime;
  end;

var
  Person: TPerson;
begin
```

في دلفي ، ومثل أية لغة أخرى تقريبا، يمكننا أن ننشئ أنواعا أخرى خاصة بنا، وهذا ما صنعناه أعلاه حيث قمنا بضم المتغيرات السابقة في هيكل واحد، ليعطينا نوعا جديدا أسميناه TPerson ، **هذا النوع الجديد هو نوع مركب، أي أنه نوع يتكون من مجموعة أنواع أخرى.**

كل تعريف داخل الهيكلية record **يسمى حقلا** Field ، **فما كان متغيرات سابقا، أصبحت داخل الهيكلية حقولا لنوع** TPerson .

ثم قمنا بإنشاء متغير جديد Person **من نوع** TPerson. **أي أنه بإستطاعته تمثيل وحمل البيانات التي يتكون منها النوع** TPerson.

والآن و بإستخدام المتغير Person يمكننا وضع القيم فيه كالتالي:

```
var
  Person: TPerson;
begin
  Person.FirstName := 'Ahmad';
  Person.LastName := 'Hamza';
  Person.BirthDate := EncodeDate(1980, 3, 15);
```

إذا، بدلا من وضع بيانات الفرد في ثلاث متغيرات متفرقة، أصبح لدينا متغيرا واحدا يحمل هذه البيانات. وهو Person ، وللوصول لكل حقل في هذا المتغير؛ نكتب اسم المتغير ثم نقطة ثم إسم الحقل: Person.FirstName تماما مثل استخدامنا للخصائص في المكونات.

طريقة عرض قيم هذا المتغير ستكون بالطريقة التالية:

```
  Canvas.TextOut(10, 10, Person.FirstName);
  Canvas.TextOut(10, 30, Person.LastName);
  Canvas.TextOut(10, 50, DateToStr(Person.BirthDate));
end;
```

حتى تكون الصورة واضحة نعيد سرد الإجرائية بالكامل:

```
procedure TForm1.Button1Click(Sender: TObject);
type
  TPerson = record
    FirstName: string;
    LastName: string;
    BirthDate: TDateTime;
  end;
var
  Person: TPerson;
begin
  Person.FirstName := 'Ahmad';
  Person.LastName := 'Hamza';
  Person.BirthDate := EncodeDate(1980, 3, 15);

  Canvas.TextOut(10, 10, Person.FirstName);
  Canvas.TextOut(10, 30, Person.LastName);
  Canvas.TextOut(10, 50, DateToStr(Person.BirthDate));
end;
```

ملاحظات سريعة:

لاحظ أن الإسم الذي أعطيناه للنوع TPerson يبدأ بحرف T وذلك للدلالة على أنه Type وهذا عرف متبع في لغة دلفي.

لاحظ أيضا أننا قبل تعريفنا لهذا النوع وضعنا التعليمة type ، وهذ أمر تتطلبه دلفي كلما أردنا إنشاء أنواعا خاصة. (حقيقة لا أعلم ما الحكمة من هذا)

ما قبل الجولة الثالثة

قبل أن ننتقل للجولة الثالثة نقترح القيام ببعض الإجراءات التنظيمية حتى لا تزدحم لدينا الأمور.

أولا:

تعريف TPerson بدلا من أن يكون محصورا في إجرائية Button1Click وخاص بها؛ نقوم بنقله إلى ملف وحدة جديدة، نسميها uOO ، حتى تتعرف عليه بقية الإجرائيات، ولفوائد أخرى سوف تتكشف لنا. وذلك كالتالي:

```
unit uOO;

interface
uses
  SysUtils;

type
  TPerson = record
    FirstName: string;
    LastName: string;
    BirthDate: TDateTime;
  end;

implementation

end.
```

كما هو واضح في السرد أعلاه، إسم الوحدة uOO ، قمنا بوضع تعريف النوع TPerson قبل قسم implementation ، كما قمنا بوضع التعليمة type قبل تعريف النوع للدلالة على أن ما يلي هذه التعليمة هي تعريفات لأنواع.

أيضا، نقوم بوضع إسم الوحدة uOO ضمن قائمة الاستخدام uses في وحدة نموذج الشاشة، كالتالي:

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms,Dialogs, StdCtrls,
  uOO;
```

الآن أصبح شكل مقدمة الإجرائية Button1Click كالتالي:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Person: TPerson;
begin
```

ثانيا:

نتقل التعليمات الخاصة بعرض البيانات ووضعها في إجرائية خاصة منفصلة. نقوم بتعريف الإجرائية تحت قسم private كالتالي:

```
  private
    procedure ShowPerson(P: TPerson);
```

ثم جسم الإجرائية:

```
procedure TForm1.ShowPerson(P: TPerson);
begin
  Canvas.TextOut(10, 10, P.FirstName);
  Canvas.TextOut(10, 30, P.LastName);
  Canvas.TextOut(10, 50, DateToStr(P.BirthDate));
end;
```

أي أن إجرائية العرض ShowPerson تستقبل محدد parameter إسمه P من نوع TPerson .

وتعليمة الإستدعاء لها كالتالي داخل Button1Click:

```

  ShowPerson(Person);
end;
```

وأصبح شكل إجرائية Button1Click كالتالي:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Person: TPerson;
begin
  Person.FirstName := 'Ahmad';
  Person.LastName := 'Hamza';
  Person.BirthDate := EncodeDate(1980, 3, 15);

  ShowPerson(Person);
end;
```

هل الأمور واضحة إلى حد الآن.

إذا لم تكن كذلك، فقد يعني هذا أنك لست معتادا على إستخدام أنواع البيانات المركبة، لذلك وقبل الدخول للجولة الثالثة، أقترح إعادة مراجعة الجولة الثانية وإستخدام أمثلة من عندك لإنشاء أنواعا أخرى باستخدام هيكلية record واستخدامها بأكثر من طريقة حتى تتمكن منها.

الجولة الثالثة

هذه هي الجولة المنتظرة.. سندخل الآن عالم المنحى للكائن..

في الجولة الأولى عبرنا عن بيانات الفرد من خلال ثلاث متغيرات بأنواع بيانات مختلفة:

```
var
  FirstName: string;
  LastName: string;
  BirthDate: TDateTime;
```

و في الجولة الثانية عبرنا عن بيانات الفرد بنوع بيانات جديد TPerson ببنية record.:

```
type

  TPerson = record
    FirstName: string;
    LastName: string;
    BirthDate: TDateTime;
  end;
```

في هذه الجولة، سنطور نوع TPerson أعلاه من بنية record إلى صنفية Class. هل قلت Class نعم Class أي أننا الآن سنتحدث بالمنحى الكائني.

كيف نحول بنية record إلى صنفية. التغيير بسيط كالتالي:

```
type

  TPerson = class(TObject)
     FirstName: string;
     LastName: string;
     BirthDate: TDateTime;
 end;
```

بدلا من تعريف TPerson على أنه بنية record ؛ عرفناه على أنه صنفية class

الآن أصبح لدينا Class. صحيح شكلها بدائي، و لكنها صنفية مكتملة، إنها صنفية بإسم TPerson ومشتقة من صنفية أخرى إسمها TObject التي تعد أصل جميع الصنفيات في دلفي.

في دلفي كل صنفية Class لابد أن تكون مشتقة من صنفية أخرى، وقد اخترنا TObject لتكون أساس الاشتقاق لصنفيتنا TPerson.

الآن سوف نرى كيفية استخدام هذه الصنفية.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Person: TPerson;
begin

  Person := TPerson.Create;

  Person.FirstName := 'Ahmad';
  Person.LastName := 'Hamza';
  Person.BirthDate := EncodeDate(1980, 3, 15);

  ShowPerson(Person);

  Person.Free;
end;
```

لقد قمنا أولا ببث الروح في المتغير Person بخلق الكائن الذي سيمثله من خلال المنهاج Create ، وقد عبرنا عنه بالأمر: Person := TPerson.Create .

المتغير Person قبل استخدامه هو مجرد جسم ميت مشكل بحسب القالب TPerson و عندما يتم إنشائه أو إحيائه بأمر Create يصير كائنا Object حيا في الذاكرة، وجاهزا للإستخدام.

عند الإنتهاء من استخدامه، و انتفاء الحاجة إليه، يجيب علينا أن ننهيه و نميته من خلال المنهاج Free، كالتالي: Person.Free

إضافة العمليات على الصنفية

الصنفية كما عرفناها حتى الآن تحوي بيانات فقط. والتي تعبر عنها الحقول التي داخل الصنفية مثل FirstName، الآن ماذا لو احتوت الصنفية على عمليات Operation أو بمصطلح آخر إجرائيات. لنجرب.

داخل الصنفية و بعد تعريف الحقول نقوم بتعريف الإجرائية أو الدالة GetFullName كالتالي:

```
TPerson = class(TObject)
  FirstName: string;
  LastName: string;
  BirthDate: TDateTime;
  function GetFullName: string;
end;
```

بعد كتابة تعريف الإجرائية بالكامل، تأكد من أن مؤشر الكتابة على سطر هذا التعريف وباستخدام لوحة المفاتيح قم بإصدار الأمر Ctrl + Shift + C . سوف تقوم دلفي ببناء جسم هذه الأجرائية. ثم نضع فيها الكود اللازم لإعطائنا إسم الفرد بالكامل.

جسم الإجرائية أو ال function سيكون كالتالي:

```
{ TPerson }

function TPerson.GetFullName: string;
begin
  result := FirstName + ' ' + LastName;
end;
```

نقوم بتعديل إجرائية العرض لدينا كي نستفيد من الدالة الجديدة.

```
procedure TForm1.ShowPerson(P: TPerson);
begin
  Canvas.TextOut(10, 10, P.FirstName);
  Canvas.TextOut(10, 30, P.LastName);
  Canvas.TextOut(10, 50, DateToStr(P.BirthDate));

  Canvas.TextOut(10, 80, P.GetFullName);

end;
```

ماذا لدينا الآ ن

لدينا الآن صنفية class إسمها TPerson ، هذه الصنفية تحوي حقول لبيانات، كما تحوي على عمليات (بالأحرى عملية واحدة). العملية ممثلة في الدالة التي إسمها GetFullName . لاحظ إن جسم الدالة خارج جسم الصنفية ولكنها تتبعها، و الدليل على ذلك إن إسم الدالة يكون مسبوقا بإسم الصنفية كالتالي:

```
function TPerson.GetFullName: string;
```

الصنفية بوضعها الحالي تعتبر صندوقا مغلقا. لماذا، لأن حقول الصنفية و بياناتها، و كذلك إجرائياتها لايمكن الوصول إليها إلا من خلال كائن ينبعث منها. أي أننا لا نستطيع أن تستخدم حقولها أو نستدعي إجرائياتها مباشرة؛ بل يجب أولا أن نقوم بتعريف متغير من نفس نوع الصنفية، ثم نقوم بخلق هذا المتغير من خلال الأمر Create ليصبح كائنا object ، عندها فقط وعبر هذا الكائن نستطيع تلمس حقول الصنفية واستدعاء اجرائياتها.

وقفة تأملية

ماهي الصنفية Class و ماهو الكائن Object وما العلاقة بينهما.

الصنفية class يمكن اعتبارها قالب، هذا القالب مشكل بحيث يحوي العناصر الي صممت له مثل البيانات (المتغيرات و الخصائص) والعمليات (مثل الإجرائيات و الدوال و المنهاجيات).

المتغير من نوع الصنفية هو جسم مستخرج من هذا القالب بعد صبه فيه. بحيث يكون شكله شكل القالب، ولكنه جسم ميت لا حياة فيه. مثل المتغير Person قبل خلقه.

الكائن Object هو الجسم السابق المستخرج من القالب لكن بعد بث الحياة فيه وخلقه من خلال Create.

خواطر وتساؤلا ت

- **من أين أتت أوامر مثل** Create و Free **؟ فهي ليست موجودة في صنفية** TPerson.

Create و Free **هي مناهج** Methods **تابعة للصنفية** TObject، **ولأن صنفية** TPerson **مشتقة منها؛ فهي بالوراثة تملك كل ما تملكه صنفية** TObject.
المنهاج Create **يسمى** Constructor **أي المشيد أو الباني الذي يقوم ببناء وتجسيد الكائن. والمنهاج** Free **يسمى** Destructor **أي الهادم أو المدمر. (في الواقع المنهاج** Free **هو مجرد إجرائية تقوم باستدعاء الهادم الحقيقي وهو** Destroy **ويمكننا استدعائه مباشرة).**

- **هل يمكن اعتبار الحقل** FirstName **خاصية** property **وهل يمكن اعتبار الدالة** GetFullName **منهاجا** Method **؟**

الحقل FirstName **الآن مجرد متغير داخل الصنفية، يمكن اعتباره شبه خاصية. اما الدالة فنعم هي منهاج أو** method.

- **ماذا يحدث لو لم نته الكائن ونتلفه؟**

سيبقى عالقا في الذاكرة محتلا لجزء منها حتى بعد إنتهاء تشغيل برنامجنا. وإذا تكرر خلق كائنات أخرى دون إتلافها، فكل عملية خلق وتجسد لكائن سيحتل حيزا إضافيا في الذاكرة، فتزداد بذلك حجم الذاكرة المستنزفة، والذي يسمى memory leakage **أي تسرب في الذاكرة.**
(في لغات أخرى مثل C# **و جافا توجد تقنية إسمها** Garbage Collection **جمع القمامة تقوم نيابة عن المبرمج بكنس وإتلاف ما علق في الذاكرة من كائنات غير مستخدمة)**

- **هل يمكن إنشاء أكثر من كائن لنفس الصنفية في نفس الوقت؟**

نعم، يمكن أن يكون لدينا الكائن Person1 **و** Person2 **وهكذا في نفس الوقت، وكل كائن منهم لديه صفاته الخاصة، هذا أحمد ، و ذاك علي. كل كائن يتم استحضاره يسمى أيضا** instance **أي حضور أو تجسد أو تمثل للصنفية التي يمثلها.**

☐

ماذا أيضا

إذا فهمت ما تم طرحه في هذه الجولة، أو إذا أصابتك ربكة أو دوخة بسيطة، أو إذا أحسست انك فهمت ولكن جزء من عقلك لايريد أن يفهم ولا يريد أن يرى الأمور بهذا المنظار الجديد؛ فتهانينا .. أنت على المسار الصحيح !!

الآن أقترح أن تتمرن قليلا على استخدام الصنفية، قم بإنشاء متغيرات أخرى من نفس نوع الصنفية TPerson، وسمّها مثلا: Person1 و Person2 ، أضف حقول أخرى على الصنفية، أضف إجرائيات ودوال أخرى، وحاول استدعائها. قم أيضا بإنشاء صنفيات أخرى، و طبق عليها اختباراتك.

نحن الآن كشفنا جزءا بسيطا من الغطاء حول مفاهيم المنحى للكائن. مفاهيم المنحى للكائن عميقة ومتشعبة، و يستحسن أن يتم الخوض فيها برفق وهدوء، فلا يتم التعمق فيها إلا بعد استيعاب ما تم فهمه وممارسته لأكثر من مرة.

في الجولة القادمة سوف نطور من الصنفية TPerson ليصبح لديها خصائص properties ومنهاجيات methods منظمة بطريقة أكثر احترافية.

الجولة الرابعة

في هذه الجولة سنغوص أكثر في مفاهيم المنحى للكائن، ونرقى بالصنفية TPerson إلى مستويات أعلى.

المنظورية

ننظر الآن إلى موضوع المنظورية visibility أو مجال الرؤية scope بالنسبة لعناصر الصنفية. أي ماهي العناصر داخل الصنفية التي يمكن للعالم الخارجي أن يراها و ينفذ إليها، و تلك التي تكون محجوبة عنه.

ماذا لو أضفنا داخل الصنفية كلمة private قبل المتغيرات مثل التالي:

```
TPerson = class(TObject)
private
  FirstName: string;
  LastName: string;
  BirthDate: TDateTime;
  function GetFullName: string;
end;
```

ثم نجرب برنامجنا. سنلاحظ أن دلفي أعطتنا رسالة خطأ عند التعليمة Person.FirstName ، **لماذا؟ لأن الكائن** Person **بعد هذا التغيير لا يستطيع أن يرى الخصائص والإجرائيات التي تم وسمها بكلمة** private **في الصنفية.**

الوسم private **تعني خاص أي خاص بالصنفية فقط، و بالتالي فإن أية تعريفات يتم سردها تحت قسم** private **لايمكن النفاذ إليها إلا فقط من قبل الاجرائيات التابعة للصنفية ذاتها.**

نفس الأمر لو استخدمنا الوسم protected **ومعناها محمي، التعريفات تحت هذا الوسم لايمكن النفاذ إليها أو رؤيتها إلا من قبل الصنفية ذاتها أو صنفية أخرى مشتقة منها (سنناقش الاشتقاق لاحقا).**

```
TPerson = class(TObject)
protected
  FirstName: string;
  ----------
```

أما إذا وضعنا الوسم public **وتعني عمومي أو عام، فإن كل ما هو تحت هذا القسم يكون مرئيا ومتاحا للعالم خارج الصنفية.**

```
TPerson = class(TObject)
public
  FirstName: string;
  ------
```

وإذا لم نقم بتحديد منظورية عناصر الصنفية، فإن دلفي افتراضيا تجعلها عمومية ومتاحة.

الخصائص

سنقوم الآن بتحويل المتغيرات في الصنفية إلى خصائص properties. **و لكن قبل أن تقوم بذلك، أقترح أن يتم إعادة ترتيب الصنفية وفق التالي:**

```
TPerson = class(TObject)
private
  LastName: string;
  BirthDate: TDateTime;
public
  FirstName: string;
  function GetFullName: string;
end;
```

أي نجعل جميع المتغيرات بالصنفية تحت قسم private ما عدا المتغير FirstName يكون تحت قسم public.

أقترح أيضا أن يتم حفظ الملف uOOP للإحتياط.

نبدأ بتحويل المتغير FirstName إلى خاصية، وذلك بكتابة التعريف property قبلها كالتالي:

```
public
  property FirstName: string;
```

الآن، نضع مؤشر الكتابة على سطر تعريف هذه الخاصية و نعطي الأمر Ctrl+Shift+C وسيقوم محرر دلفي آليا بتكملة التعريف وتوليد الكود اللازم. شكل الصنفية سيكون كالتالي:

```
TPerson = class(TObject)
private
  LastName: string;
  BirthDate: TDateTime;
  FFirstName: string;
  procedure SetFirstName(const Value: string);
public
  property FirstName: string read FFirstName write SetFirstName;
  function GetFullName: string;
end;
```

```
procedure TPerson.SetFirstName(const Value: string);
begin
  FFirstName := Value;
end;
```

لقد قام المحرر بتوليد تعليمات إضافية آليا والتي تمثلت في الآتي:

- إضافة متغيرا جديدا أسماه FFirstName أي بنفس اسم الخاصية مسبوقا بحرف F وجعل نوعها من نفس نوع الخاصية أي string.
- إضافة إجرائية بنفس إسم الخاصية مسبوقة بكلمة Set وهي SetFirstName مع محدد parameter بنفس نوع الخاصية.
- إضافة جسم الإجرائية SetFirstName والتي يتم فيها تخصيص القيمة المررة لها للمتغير FFirstName .

- في سطر تعريف الخاصية قام بتحديد مصدر قراءة قيمة الخاصية read ومصدر تحديد قيمة الخاصية write . وفي حالتنا هذه فإن قيمة الخاصية يستمدها من المتغير FFirstName ، كما أنها تحدد بواسطة الإجرائية SetFirstName .

إذا وجدت كلامي السابق غامضا أو مبهما ، فلا تقلق، سيتم توضيحه أكثر بمزيد من الأمثلة.

يمكن الآن تكرار العملية السابقة مع باقي المتغيرات، ننقل تعريف المتغيرات LastName و BirthDate إلى قسم public ، ثم نضع مؤشر الكتابة على سطر إحدى الخصائص وتنفيذ الأمر Ctrl+Shift+C ليقوم المحرر بتوليد التعليمات المتعلقة بهذه الخصائص.

شكل الصنفية سيكون كالتالي:

```
  TPerson = class(TObject)
  private
    FFirstName: string;
    FLastName: string;
    FBirthDate: TDateTime;

    procedure SetBirthDate(const Value: TDateTime);
    procedure SetFirstName(const Value: string);
    procedure SetLastName(const Value: string);
  public
    property FirstName: string read FFirstName write SetFirstName;
    property LastName: string read FLastName write SetLastName;
    property BirthDate: TDateTime read FBirthDate write SetBirthDate;

    function GetFullName: string;
  end;
```

وهذه مجموعة ال Setters التابعة لهذه الخصائص:

```
procedure TPerson.SetFirstName(const Value: string);
begin
  FFirstName := Value;
end;

procedure TPerson.SetLastName(const Value: string);
begin
  FLastName := Value;
end;

procedure TPerson.SetBirthDate(const Value: TDateTime);
begin
  FBirthDate := Value;
end;
```

إضافة خاصية جديدة

الآن نقوم بتحديد خاصية جديدة. الخاصية ستكون بإسم Age أي العمر، وغرضنا منها أن تخبرنا بعمر الفرد من واقع معلومة تاريخ الميلاد. نريد أيضا من الخاصية أن تكون للقراءة فقط، أي أنها تعطي عمر الفرد فقط ولا تقبل أن يتم تحديد قيمة هذا العمر من خارج الصنفية.

```
public
  ..........
  ..........
  property Age: integer read GetAge;
```

لاحظ أنه في تعريف الخاصية حددنا read فقط و لم نحدد write .

إذن حساب العمر سيكون من الدالة التي أسميناها GetAge كما هو واضح أعلاه. يمكننا تعريف هذه الدالة وبناء جسمها يدويا، أو نترك محرر دلفي يقوم بهذه العملية.

تعريف الدالة GetAge سيكون تحت قسم private كالتالي:

```
private
  ..........
  ..........
  function GetAge: integer;
```

أما جسم الدالة GetAge سيكون كالتالي:

```
function TPerson.GetAge: integer;
begin
  result := Round(YearSpan(Now, BirthDate));
end;
```

الدالة تقوم بإعطاء العمر بصورة تقريبية، باستخدام الدالة YearSpan . (يجب إضافة DateUtils إلى قسم uses).

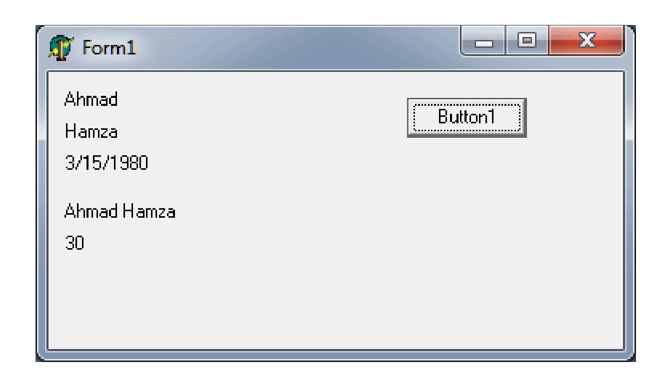
لاحظ أن الدالة GetAge معرفة في قسم private وبالتالي فإنها لن تكون مرئية للعالم الخارجي.

الآن لنستخدم هذه الخاصية الجديدة في شاشة العرض لدينا.

```
procedure TForm1.ShowPerson(P: TPerson);
begin
   Canvas.TextOut(10, 10, P.FirstName);
   Canvas.TextOut(10, 30, P.LastName);
   Canvas.TextOut(10, 50, DateToStr(P.BirthDate));

   Canvas.TextOut(10, 80, P.GetFullName);
   Canvas.TextOut(10, 100, IntToStr(P.Age));
end;
```

لنرى الآن شكل شاشة برنامجنا



المزيد من التوضيح

عندما كانت FirstName مجرد متغير في الصنفية، يمكن لأي مستخدم للصنفية أن يقرأ قيمتها أو يكتب لها قيمة جديدة دون قيود. فيقول Caption := Person.FirstName أو يقول Person.FirstName := 'Ali'.

بعدما حولنا هذا المتغير إلى خاصية property، أصبح لدينا القدرة لأن نحدد قيودا تحكم عمليات القراءة والكتابة لهذه الخاصية. كيف ذلك؟ الخاصية تعطينا الخيارات التالية للتحكم:

أي مستخدم للصنفية عندما يريد معرفة قيمة الخاصية أي قرائتها فإن الصنفية تعطيه القيمة حسب التالي:

```
property FirstName: string read FFirstName;
```

الخاصية تعطيه القيمة الموجودة في المتغير FFirstName.

```
property FirstName: string read GetFirstName;
```

الخاصية تعطيه القيمة الناتجة عن دالة إسمها GetFirsName (لم نحددها في مثالنا السابق، ولكن في الخاصية Age حددنا دالة تعطي القيمة)

أي مستخدم للصنفية عندما يريد أن يحدد قيمة الخاصية أي كتابتها فإن الصنفية تسمح بذلك حسب التالي:

```
property FirstName: string read FFirstName write FFirstName;
```

الخاصية تأخذ القيمة وتضعها مباشرة في المتغير FFirstName

```
property FirstName: string read FFirstName write SetFirstName;
```

الخاصية تأخذ القيمة وتمررها أولا إلى الإجرائية SetFirstName التي ستقرر ماذا ستفعل بها.

```
property FirstName: string read FFirstName;
```

الخاصية لا تقبل أية قيمة لها، أي أنها للقراءة فقط read only ترفض أية قيمة تعطى لها.

```
property FirstName: string read GetFirstName write SetFirstName;
```

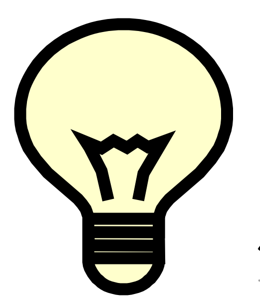
عند القراءة تعطي ناتج الدالة GetFirstName وعند الكتابة تأخذ القيمة وتمررها إلى الإجرائية SetFirstName.

إذن هذه هي خيارات تحديد عمليات القراءة و الكتابة للخاصية، أي كيفية قراءة قيمة الخاصية وكيفية تحديدها. عمليات القراءة و الكتابة للخاصية تسمى Getters و Setters .

عندما نطلب من محرر دلفي إكمال بناء كود الخاصية بواسطة الأمر Ctrl+Shift+C يقوم بتوصيف ال Getter على أنه متغير من نفس نوع الخاصية مثل FFirstName، ويقوم بتوصيف ال Setter على أنه إجرائية بمحدد من نفس النوع مثل SetFirsName . هذه هي الوضعية السائدة في أغلب الأحيان، إذا أردت وضعية أخرى يمكنك تغيير ذلك.

في ال Setter مثل إجرائية SetFirstName نرى أن الإجرائية توجد بها تعليمة واحدة و هي :

FFirstName := Value و هي التعليمة الوحيدة التي يضعها محرر دلفي، و التي تحتاجها الخاصية لمعرفة قيمتها .



ملاحظة

هل لاحظتم أخوتي أن المتغيرات التابعة للخصائص مثل FFirsName و FLastName تبدأ بحرف F ؟ طبعا نستطيع أن نسمي هذا المتغيرات بالطريقة التي نريدها، ولكن وكعرف عام أيضا في دلفي؛ تكون هذه المتغيرات مسبوقة بحرف F للدلالة على أنها Field أي حقل داخل الصنفية.

شيء في الحلق

يوجد شيء في حلقي يزعجني منذ بدئنا للجولة الثالثة. ولم أشأ أن أشير إليه حتى لا أشوش أو أربك مسار تفكيرنا وفهمنا للأمور، و لكن الآن و بعد أن استوفينا جولتنا الرابعة، أظن أن الوقت قد حان لأن أفصح عنه.

تذكرون سادتي أنه عندما قمنا بإنشاء أو خلق الكائن Person فعلنا ذلك من خلال التعليمة:

```
Person := TPerson.Create;
```

وكما تذكرون أيضا، فقد أشرنا إلى أن الكائن الذي ينشأ يجب أن يتم إتلافه بعد الانتهاء من استخدامه.

OK، كقاعدة ذهبية : عند خلق أي كائن يجب أن يغلّف الكود الخاص بالتعامل مع هذا الكائن ضمن حائط try...finally وذلك كالتالي:

```
Person := TPerson.Create;

try
  Person.FirstName := 'Ahmad';
  Person.LastName  := 'Hamza';
  Person.BirthDate := EncodeDate(1980, 3, 15);

  ShowPerson(Person);

finally
  Person.Free;
end;
```

بهذا نضمن أنه إذا حدث أي خطأ عند استخدامنا للكائن Person فإن برنامجنا لن يقفز من مكان الخطأ إلى خارج الإجرائية؛ بل يستمر ويذهب إلى التعليمات تحت قسم finally ليقوم بإزالة هذا الكائن ويحرر الذاكرة منه. سواء حدث خطأ أم لم يحدث.

شيء من الفلسفة

بعد أن أخذنا و بطريقة برمجية فكرة عن مفاهيم المنحى للكائن، لنسرد هنا بعضا من المصطلحات الخاصة بهذا المجال. ولتعذرني أخي القارئ فسوف يكون في حديثنا بعض الفلسفة، فأرجو منك أن تخلع T-Shirt البرمجة و تلبس عباءة الفلسفة، و تحمل معي قليلا.

Entity

بمعنى كينونة، أو كيان أو وجود ويقصد به أي شيء في غير عالم البرمجة سواء كان هذا الشيء مادي أو معنوي. في مثالنا السابق كان الأستاذ للهأحمد حمزةلله هو إنسان بشحمه و لحمه يأكل و يتنفس و يمشي في الأسواق، مثله مثل غيره ممن هو إنسان، إذن كلمة إنسان تعبر عن مفهوم تجريدي يجمع كل من هو بني آدم ، فيمكن القول أن *أحمد حمزة* هو entity كينونة، وأن إنسان كينونة أيضا لكن بمفهوم تجريدي أعلى.

Class

بمعنى صنفية، وهي إطار تجريدي يضم الكينونات التي نرى أنها تتشابه في سماتها و تصرفاتها، الأخ للهأحمد حمزةلله رأيناه يشبه في شكله و تصرفاته أفرادا آخرين مثل للهعليلله و للهإسماعيلله، عليه يمكن أن نضمهم تحت مسمى واحد و هو إنسان أو شخص، فأحمد إنسان وعلي إنسان وإسماعيل كذلك، هذا المُسمّى الموحد الذي يجمع بينهم نحدده أكثر برمجيا ونقول عنه صنفية class ونسمي هذه الصنفية TPerson ، و نحدد في هذه الصنفية العناصر التي نرى أنها تجمع بين هؤلاء الكينونات أو التي تشكل مفهوم إنسان، في مثالنا السابق كانت رؤيتنا ضيقة، وحددنا فقط ثلاثة أو أربعة عناصر تجمع بين هؤلاء الأفراد أو يضمها مفهوم إنسان مثل FirstName و LastName ، ووضعناها داخل هذه الصنفية.

Object

وتعني كائن أو ماهية، و الكائن هنا هو وجود برمجي، فبعد ما رسمنا إطارا لصنفية TPerson ، نقوم باستحضار أو تجسيد كائن برمجي من نفس الفصيلة أو الصنفية فنقول Person := TPerson.Create ثم حددنا أن هذا الكائن البرمجي إسمه الأول "أحمد" وإسمه الثاني للهحمزةلله. بحسب الخصائص أو العناصر التي أطرناها في الصنفية. الكائن Person الذي إسمه الأول للهأحمدلله وإسمه الثاني للهحمزةلله هو كائن برمجي، من فصيلة أو صنفية TPerson يعيش في عالمنا البرمجي أي برنامجنا، وأردنا به أن يمثل السيد للهأحمد حمزةلله الحقيقي الذي يعيش في أرض الواقع.

أخذ و رد

من خلال قسم المجلة في منتدى دلفي للعرب يمكنكم إخوتي الكرام طرح أية استفسارات تخص ماورد في هذا المقالة لمناقشتها سويا.

إذا أحسست أثناء قرائتك لهذا المقال بوجود أي غموض أو لبس أو عدم وضوح كاف، أو إطالة زائدة، يرجى الإشارة إلى ذلك، و تحديد الموضوع أو الجزء الذي يعاني من أي عيب حتى يتم تعديله و تحسينه. أيضا إذا كان لديك أي اقتراح أو ملاحظة تشعر أنها ستسهم في زيادة توضيح المفاهيم التي طرحت في هذه المقالة فلا تتردد أخي في طرحها.

أيضا إذا وجدت أية أخطاء ضمن طرحنا لمفاهيم المنحى للكائن؛ فإني أرجوكم أن تقوموا بالتنبيه إليها حتى يتم تصحيحها فورا.

مرفق مع المقالة الكود البرمجي الذي استخدمناه في هذه المقالة مقسم حسب كل جولة.

الجولة القادمة

الجولات القادمة ستكون في الجزء الثاني من هذه المقالة إن شاء اللّه، و فيها سنتوسع في الحديث عن الصنفيات، من ذلك كيفية إنشاء صنفية مشتقة من صنفية أخرى، وخيارات لإستخدام الخصائص والمناهج، كذلك كيفية حفظ واستراجاع البيانات الخاصة بالكائنات في برنامجنا.

شيء من الصور

الآن أتـركـكم مع بعض الصور التي تـلخص بعض ما تم الحديث عنه حول مفاهيم المنحى للكائن.



```
TPerson = Class
```

الصنفية TPerson عبارة عن قالب

```
Var
   Person: TPerson
```

المتغير Person من نوع الصنفية TPerson
جسم ميت



```
Person := TPerson.Create
```

Person صار كائنا حيا Object له مكان خاص به في الذاكرة



```
Person.FirstName := 'xxxx'
```

الكائن Person بعد مزيد من التحديد لخصائصه

```
Person.Free
```

الكائن Person بعد إماتـته وعودته لمجرد متغير

نسخ الملفات في دلفي

في حالات كثيرة، قد تحتاج في برنامجك أن تقوم بنسخ ملفات من موضع لآخر...

لنر كيف (وبكم طريقة؟) يمكن القيام بذلك من خلال Delphi

1. استخدام تابع API معرف باسم CopyFile كما يلي:

```
function CopyFile(lpExistingFileName, lpNewFileName: PChar;
bFailIfExists: BOOL): BOOL; stdcall;
```

مثال:

```
if CopyFile('c:\windows\explorer.exe', 'd:\explorer.exe', False) then
ShowMessage('OK')
else
ShowMessage('Failed');
```

تتطلب الدالة كتابة المسار الكامل للملفين. المعلمة الثالثة (bFailIfExists) تعني إخفاق الدالة حالة وجود الملف.

يمكنك إعادة صياغة تابع خاص بك لتسهيل الاستخدام، مثال:

```
function MyCopyFile(const Source, Dest: string; Overwrite: Boolean =
False): Boolean;
begin
Result := CopyFile(PChar(Source),
PChar(IncludeTrailingPathDelimiter(Dest)+
ExtractFileName(Source)), Overwrite);
end;
```

2. باستخدام إجراءات التحكم في الملفات (AssignFile /BlockRead /BlockWrite ...) يمكن نسخ الملفات بشكل يدوي والحصول على سرعة أكبر، مع إضافة مكون TProgressBar لتوضيح تقدم العملية، مثال:

```
Type
TCallBack=procedure (Position,Size:Longint);
```

```
procedure CallBackProc(Position, Size: LongInt);
begin
fMain. pbCopyProgress.Max := Size;
fMain. pbCopyProgress.Position := Position;
end;
```

```
procedure FastFileCopy(Const InfileName, OutFileName: String; CallBack:
TCallBack);
Const BufSize = 3*4*4096;
Type
PBuffer = ^TBuffer;
TBuffer = array [1..BufSize] of Byte;
var
Size              : integer;
Buffer            : PBuffer;
InFile, OutFile   : File;
SizeDone,SizeFile: Longint;
begin
if (InFileName <> OutFileName) then
begin
buffer := Nil;
AssignFile(InFile, InFileName);
System.Reset(InFile, 1);
try
SizeFile := FileSize(InFile);
AssignFile(OutFile, OutFileName);
System.Rewrite(OutFile, 1);
try
SizeDone := 0; New(Buffer);
repeat
BlockRead(InFile, Buffer^, BufSize, Size);
Inc(SizeDone, Size);
CallBack(SizeDone, SizeFile);
BlockWrite(OutFile,Buffer^, Size)
until Size < BufSize;
FileSetDate(TFileRec(OutFile).Handle,
FileGetDate(TFileRec(InFile).Handle));
finally
if Buffer <> Nil then Dispose(Buffer);
System.close(OutFile)
end;
```

```
finally
System.close(InFile);
end;
end else
Raise EInOutError.Create('File cannot be copied into itself');
end;
```

الاستخدام بنفس الطريقة (كتابة المسار الكامل للملفين).

3. نفس الطريقة السابقة باستخدام النمط TStreamFile، مثال:

```
Procedure CopyStreamFile(InFileName,OutFileName:String);
Const
BufSize= 3*4*4096;
Var
InFile,OutFile:TStream;
Buffer:Array[1..BufSize] Of Byte;
ReadBufSize:Integer;
Begin
InFile := nil; OutFile := nil;
Try
InFile:=TFileStream.Create(InFileName,fmOpenRead);
OutFile:=TFileStream.Create(OutFileName,fmCreate);
Repeat
ReadBufSize:=InFile.Read(Buffer,BufSize);
OutFile.Write(Buffer,ReadBufSize);
Until ReadBufSize<>BufSize;
Finally
InFile.Free;
OutFile.Free;
End;{Try}
End;{CopyFile}
```

4. طريقة أخرى بتنفيذ تعليمات سطر الأوامر بسهولة، مثال:

```
function CopyFileCmd(Source, Dest: string): Boolean;
begin
Result := WinExec(PChar(Format('cmd copy %s %s',[Source, Dest])),
SW_HIDE) > 31;
end;
```

والاستخدام سهل:

```
if MyCopyFile('c:\windows\explorer.exe', 'd:\') then
ShowMessage('OK');
```

5. **طريقة أخرى باستخدام التابع** SHFileOperation **المكتبة** ShellAPI **المعرف في** shell32، **حيث يمكن عرض واجهة نسخ الملفات وتقدم العملية**:



المثال:

```
uses
ShellAPI;

function CopyFileOrFolder(Source, Destination: string): boolean;
var
SHFileOp: TSHFileOpStruct;
Error: integer;
Aborted: boolean;
begin
FillChar(SHFileOp, SizeOf(SHFileOp), 0);
SHFileOp.Wnd := Application.Handle;
SHFileOp.wFunc := FO_COPY;
SHFileOp.pFrom := PChar(Source + #0);
SHFileOp.pTo := PChar(Destination + #0);
SHFileOp.fFlags := FOF_ALLOWUNDO;
Error := SHFileOperation(SHFileOp);
Aborted := (Error = 117);
if Aborted then
Error := 0;
Result := (Error = 0) and not Aborted;
end;
```

تسمح هذه الطريقة بنسخ أي ملف أو مجلد.

6. آخر طريقة نعرضها هنا باستخدام تابع API جديد بالنسبة إلى إصدارات ويندوز السابقة

يستخدم التابع CopyFileEx سمات إضافية ويسمح بالتحكم في عملية النسخ ومشاهدة تقدم العملية...

لنشاهد تعريف التابع:

```
BOOL CopyFileEx(

LPCWSTR lpExistingFileName,        // pointer to name of an existing
file
LPCWSTR lpNewFileName,      // pointer to filename to copy to
LPPROGRESS_ROUTINE lpProgressRoutine,       // pointer to the callback
function
LPVOID lpData,      // to be passed to the callback function
LPBOOL pbCancel,    // flag that can be used to cancel the operation
DWORD dwCopyFlags   // flags that specify how the file is copied
);
```

يمكننا من خلال التعريف صياغة التابع في دلفي بمثل هذه الطريقة:

```
private
FCancelled: Boolean;
...
```

```
function TfMain.CopyWithProgress(sSource, sDest: string; Overwrite:
Boolean = True): Boolean;
function CopyProgressRoutine(TotalFileSize,
TotalBytesTransferred,
StreamSize,
StreamBytesTransferred: LARGE_INTEGER;
dwStreamNumber,
dwCallbackReason: DWORD;
hSourceFile,
hDestinationFile: THandle;
lpData: Pointer): DWORD; stdcall;
begin
// just set size at the beginning
if dwCallbackReason = CALLBACK_STREAM_SWITCH then
TProgressBar(lpData).Max := TotalFileSize.QuadPart;

TProgressBar(lpData).Position := TotalBytesTransferred.QuadPart;
Application.ProcessMessages;
Result := PROGRESS_CONTINUE;
end;
begin
// set this FCancelled to true, if you want to cancel the copy operation
FCancelled := False;
```

```
Result := CopyFileEx(PChar(sSource),
PChar(IncludeTrailingPathDelimiter(sDest)+
ExtractFileName(sSource)), @CopyProgressRoutine, pbCopyProgress,
@FCancelled,
COPY_FILE_FAIL_IF_EXISTS     * Ord(Overwrite));
end;
```

يمكن إلغاء عملية النسخ بضبط المتغير FCancelled على القيمة True، مثال الاستخدام:

```
if CopyWithProgress(edFileToCopy.Text, edDest.Text) then
ShowMessage('OK')
else
ShowMessage('Failed');
```

الأمثلة السابقة في مرفقات المجلة.



مقدمة في برمجة الشبكات باستخدام INDY

للبدء في هذا المجال سنقوم بشرح بعض المصطلحات الأساسية

مصطلحات الشبكات

TCP/IP

إختصار ل Transmission Control Protocol and Internet Protocol.

و يعني في أغلب الأحيان بروتوكول للشبكة.

Client

و يعني عميل.

العميل باختصار هو عبارة عن العملية التي تبدأ بالاتصال بالخادم كما هو الحال في الهاتف فالشخص الذي يبدأ المكالمة هو العميل.

Server

هو الخادم.

الخادم هو العملية التي تستقبل الطلبات من العملاء و الخادم قادر على تلقي أكثر من طلب في نفس الوقت و كل طلب أو اتصال قادم للخادم هو من عميل.

IP Address

اختصار ل Internet Protocol Address.

كل جهاز على شبكة TCP/IP **لديه عنوان خاص على هذه الشبكة ، أي هذا العنوان مرتبط بهذا الجهاز. و ال** IP **يكون** 32 **بت (إذا كان الاصدار الرابع) و هذا العنوان يشار له بأرقام و نقاط مثلا** 192.168.1.1

ال IP **هو عبارة عن رقم الهاتف في منظومة الاتصال و بما أن المنزل مثلا يمكن أن يحوي أكثر من رقم هاتف فهذا يعني أن كل جهاز يمكن أن يكون له أكثر من** IP .

Port

المنفذ و هو عبارة عن رقم (يأخذ الاحتمالات من 0 **إلى** 65535

يشبه المنفذ باللاحقة التي تسبق رقم الهاتف فمثلا إذا اردت الاتصال بالسعودية فانت تطلب الرقم *00966* **و هذا الرقم يوجهك إلى السعودية أما إذا اردت الاتصال بسوريا مثلا فأنت تطلب الرقم** 00963 **و هكذا فالمنفذ هو الموجه لعملية الاتصال و هناك منافذ معروفة مسبقا و تستخدم بالتطبيقات فمثلا المنفذ** 80 **يستخدم في** Http **و** 21 **يستخدم في ال** FTP.

و المنافذ دون 1024 **هي مستخدمة مسبقا لذلك وجب علينا ألا نستخدم هذه المنافذ إلا بحال اردنا الاتصال بهذه البروتوكولات المعروفة فمثلا نريد صفحة من الانترنت فإننا نتصل بالبروتوكول** Http **عبر المنفذ** 80.

TCP

اختصار ل Transmission Control Protocol.

يشار إليه أحيان إلى بروتوكول التدفق (Stream Protocol)

TCP/IP **يحوي الكثير من البروتوكولات و وسائل لنقل البيانات و أشهر هذه النواقل هي** TCP **و** UDP.

ال TCP **هو عبارة عن بروتوكول معتمد على الاتصال ؛ و هذا يعني أنه ليتم نقل البيانات عبر هذا البروتوكول يجب علينا أن نكون متصلين بخادم و بعدها نرسل المعلومات.**

هذا النوع من البروتوكولات يضمن وصول البيانات حالما ترسل عبره ، و هو يضمن دقة عملية النقل ، أي لا يوجد ضياع بالبيانات عبره.

UDP

اختصار ل User Datagram Protocol.

هذا النوع من البروتوكولات يستخدم لنقل كتل البيانات و لا يتطلب وجود اتصال مع الخادم لتتم عملية الارسال.

فهو يسمح بنقل الحزم الصغيرة من البيانات للجهاز دون وجود اتصال بدئي.

و لكن هذا النوع من البروتوكولات لا يضمن وصول البيانات للهدف و ربما لا تصل بنفس الترتيب الذي ارسلت به . و عندما يتم ارسال حزمة من البيانات عبر هذا البروتوكول فإنها ترسل ككتلة واحدة لذلك يجب علينا ألا تتجاوز أكبر حجم للحزمة المحدد بواسطة مكدس TCP/IP .

طبعا نظرا لهذه الاعتبارات نظن أن هذا البروتوكول لا يجدي نفعا و لكن الكثير من بروتوكولات الدفق (Streaming Protocols) **تستخدم هذا البروتوكول مثل** RealAudio

وثوقية حزم ال UDP **تعتمد على وثوقية نوع الشبكة المستخدمة؛ فمثلا الشبكات المنزلية من نوع** Lan **تعتبر ذات وثوقيه عالية و العديد من البرامج تستخدم هذا البروتوكول.**

طبعا ال UDP **يمكن أن تستخدم و لكن يجب دائما أن نضع بالحسبان أن البيانات ربما لن تصل لهدفها المنشود.**

و لأن ال UDP **لا يوجد لديه تأكيد لعملية وصول البيانات لذلك فهو لا يضمن عملية الوصول ؛ أي إذا ارسلت حزمة من البيانات لجهاز ما ففي هذه الحالة لا يوجد لدينا أي طريقة لمعرفة هل فعلا وصلت المعلومات لهذا الجهاز أم لا، أي المكدس لا يستطيع تحديد ذلك و لذلك فهو لن يرسل خطأ بحال عدم الوصول ، و لذلك إذا اردنا أن نعلم إذا وصلت المعلومة أم لا علينا تزويد البرنامج بآلية تعلم الجهاز المرسل أنه وصلت المعلومة التي ارسلها.**

ICMP

اختصار ل Internet Control Message Protocol

هذا البروتوكول عبارة عن بروتوكول تحكم و صيانة. بالحالة الاعتيادية نحن لا نستخدم هذا البروتوكول.

هذا البروتوكول يستخدم لعملية الاتصال مع الموجهات (Routers) **و أجزاء الشبكة الأخرى فهو يتيح لعقد الاتصال بمشاركة معلومات عن حالة** IP **معين و مشاركة اعلامات الخطأ. فهو يستخدم مثلا في ال** Ping **و** TraceRoute **و غيرها من البروتوكولات.**

Thread

هو عبارة عن المسار التنفيذي للبرنامج. الكثير من البرامج لديها Thread **واحد و لكن يمكن انشاء العديد من الخيوط** (Threads) **لإنشاء مسارات تنفيذية منفصلة لبرنامج ما.**

في انظمة تحوي العديد من المعالجات فإن ال Threads **تتوزع بين هذه المعالجات لكي يكون هناك اقلاع اسرع. أما بالنسبة للأنظمة ذات المعالج الوحيد فإن هذه ال** Threads **يمكنها أن تنفذ بالاستباق.**

Fork

مصطلح يعني تفرع أو تشعب.

انظمة ال Unix **لا تدعم ال** Threading **لكنها تدعم بدلا عن ذلك ال** Fork.

ال Threading **هو عبارة عن خط أو مسار جديد منفصل يتكون و لكن هو لا يزال ضمن نفس العملية** (Process) **و لهذا فهو يتشارك نفس مساحة الذاكرة.**

أما ال Forking **فهو عبارة عن انقسام العملية** (Process) **نفسها اي يتم انشاء عملية جديدة و تمرر لها المقابض**(Handles).

ال Forking **ليس فعال مثل ال** Threading **و لكن لديه نفس محاسن ال** Threading **إضافة لذلك فهو (أي ال** Forking) **أكثر استقرارا و هو اسهل للبرمجة في العديد من الحالات.**

Winsock

اختصار ل Windows Sockets

هو عبارة عن مجموعة توابع API **معرفة و معيارية لبرمجة بروتوكولات الشبكة. و أكثر ما تستخدم لبرمجة ال** TCP/IP **و لكن يمكن أن تستخدم لبرمجة بروتوكولات شبكة أخرى مثل** (IPX/SPX) Novell.

ال Winsock **يمكن الوصول لها عبر ملفات ربط ديناميكية** (Dll) **و هي جزء من نظام** Windows.

Stack

مصطلح يشير إلى طبقة من نظام التشغيل التي تنفذ الشبكة و توفر ال API **للمطورين ليستطيعوا الوصول للشبكة و في نظام** Windows **المكدس** (Stack) **ينجز ب** Winsock.

مقدمة في INDY

لنبدأ أولا بالمصطلح، فمصطلح Indy **اختصار ل** Internet Direct

تم تصميم الاندي منذ البداية ليكون متعدد الخيوط (Threadable) **و الطريقة المتبعة في انشاء الخادم و العميل في الاندي مشابهة لما يتم في انظمة ال** Unix **و لكن بصورة أسهل بكثير لأنك تعمل مع اندي و دلفي ببساطة.**

بشكل نموذجي خوادم يونكس (Unix) **لديها أكثر من عملية مصغية تنتظر الطلبات من العملاء و تقوم بعملية خدمة لطلبات هؤلاء العملاء و يتم بشكل طبيعي انشاء فرع** (Fork) **ليتم التعامل مع كل عميل على حدى و هذا مما يجعل الامور سهلة فكل** Fork **يتعامل مع عميل واحد فقط لا غير. و العملية لديها سياق الأمن الخاص بها و هذا السياق يُعد بواسطة العملية الاساسية التي قامت بإنشاء ال** Fork **بناء على الاعتمادات أو التوثيق أو غيرها من الوسائل.**

خوادم Indy **تعمل بنفس الطريقة السابقة و لكن ال** Windows **لا يملك ال** Fork **بل يملك ال** Thread **و خوادم الاندي تقوم بتخصيص** Thread **لكل عميل.**

خوادم الاندي تقوم بإعداد thread مصغي جديد منفصل عن ال Thread الأساسي للبرنامج و لذلك يقوم ال Thread المصغي بعملية اصغاء لطلبات العميل.

لذلك يتم توليد Thread جديد لخدمة طلبات العميل و يتم بعدها تفعيل حوادث في سياق أو محيط هذا ال Thread (Context Thread).

منهجية اندي

الاندي مختلف عن مكونات المقابس (Socket Components) التي نعرفها. و ستجد أن التعامل مع اندي سهل للغاية.

تقريبا كل المكونات الأخرى تستخدم الاستدعاءات الغير كتلية (Non-Blocking) و تعمل بشكل غير متزامن (asynchronously) و هذه المكونات تتطلب منا الاستجابة لكل الحوادث و اعداد حالة الاجهزة و غالبا ما تطلب حلقات انتظار.

كمثال في المكونات الأخرى عندما نستدعي حدث الاتصال (Connect) فيجب علينا الانتظار ليتم تفعيل حدث الاتصال أو نفعل حلقة حتى تشير لدينا خاصية أنه تم الاتصال.

أما في اندي فإنه عندما نستدعي الاتصال فانه ننتظر نتيجة هذا الاستدعاء فإذا تم النجاح فهذا يعني أنه المهمة أكملت بنجاح و إلا فإن استثناء (Exception) سيظهر.

التعامل مع اندي يشبه التعامل مع الملفات فاندي تسمح لنا بوضع كل الشفرات (Code) في مكان واحد بدلا من بعثرتها في احداث مختلفة.

اندي مصمم ليعمل جيدا مع الخيوط (Threads).

لماذا الاندي مختلف

- الاندي يستخدم توابع API للمقابس الكتلية (Blocking Socket API)
- الاندي لا يعتمد على الاحداث و لكن الاندي لديه أحداث تستخدم للأغراض التعليمية و نحن لسنا بحاجة لها.

- الاندي مصمم ل Threads و لكن الاندي يمكنه أن يستخدم بدون Threads.
- الاندي مبرمج بواسطة البرمجة التسلسلية (Sequential Programming).
- الاندي لديه مستوى عالي من التجريد (Abstraction) فمعظم المكونات الأخرى لا تعزل المبرمج عن المكدس بصورة فعالة و بدلا من ذلك تقوم بتمرير تعقيدات المكدس إل مجمع الدلفي أو C++

نظرة على العملاء

الاندي مصمم ليوفر درجة عالية من التجريد فتعقيدات و تفاصيل المكدس الخاص ب TCP/IP مخفية عن المبرمج بالاندي.

فمثلا جلسة العميل تبدو:

```
with IndyClient do begin
  Host := 'postcodes.atozedsoftware.com'; // Host to call
  Port := 6000; // Port to call the server on
  Connect;
 Try
 // Do your communication
 finally
  Disconnect;
 end;
end;
```

نظرة على الخوادم

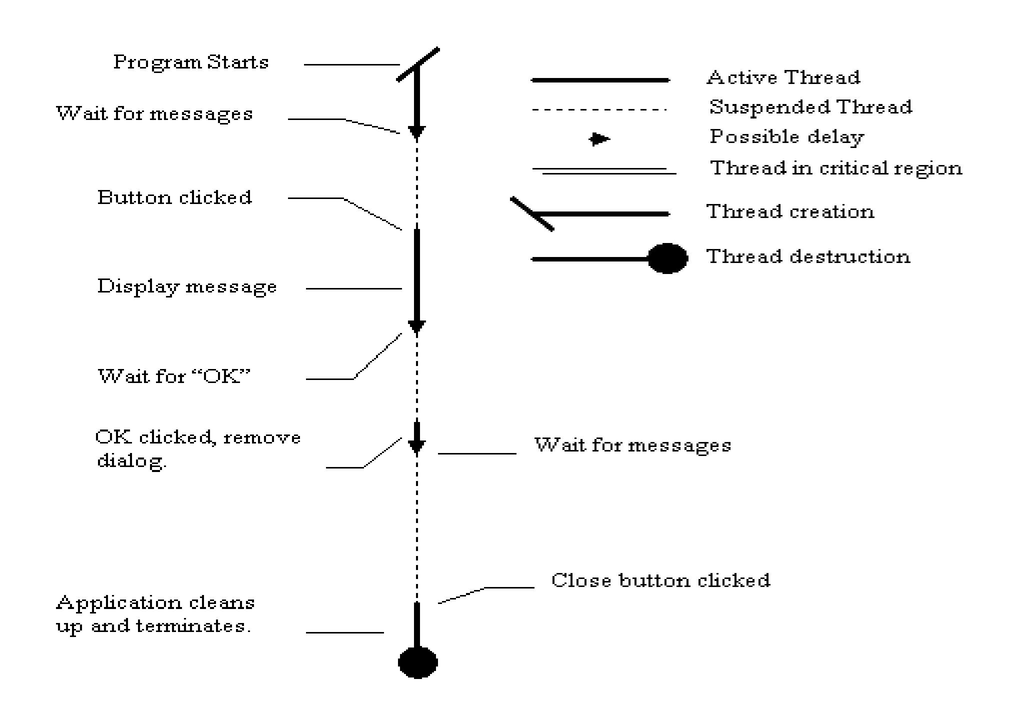
تقوم خوادم الاندي بإنشاء Thread مصغية منفصلة عن Thread الاساسي للبرنامج و هذا ال Thread المصغي يقوم بالإصغاء لطلبات العميل و هكذا يقوم خادم الاندي بتوليد خيط يصغي و يخدم العميل .

Threading

هو عملية استخدام الخيوط (Threads) لتنفيذ المهام و الاندي يستخدم Threading بشكل كبير في خوادمه و هذا مفيد للخادم و العميل.

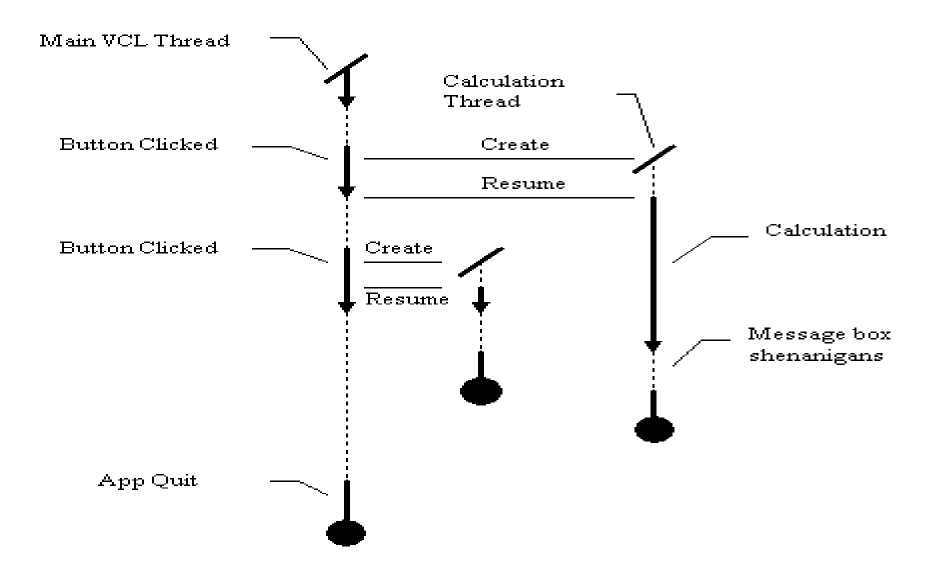
المقابس الغير كتلية (Non-Blocking Sockets) يمكن أن تلجأ إلى الخيوط و لكنها تحتاج للمزيد من المعالجة و مزاياها ستُفقد في المقابس الكتلية (Blocking Sockets).

لكي نفهم ال Threads بشكل معمق لننظر إلى هذه الصورة



البرنامج أثناء تنفيذه للعمليات يسير في خط واحد كما هو موضح بالصورة، و يكون بانتظار الرسائل من النظام. فعند الضغط على زر معين يقوم البرنامج بتنفيذ أوامر هذا الزر. في حالتنا هذه البرنامج مكون من خيط واحد أي لا يوجد لديه سوى Thread واحد فقط.

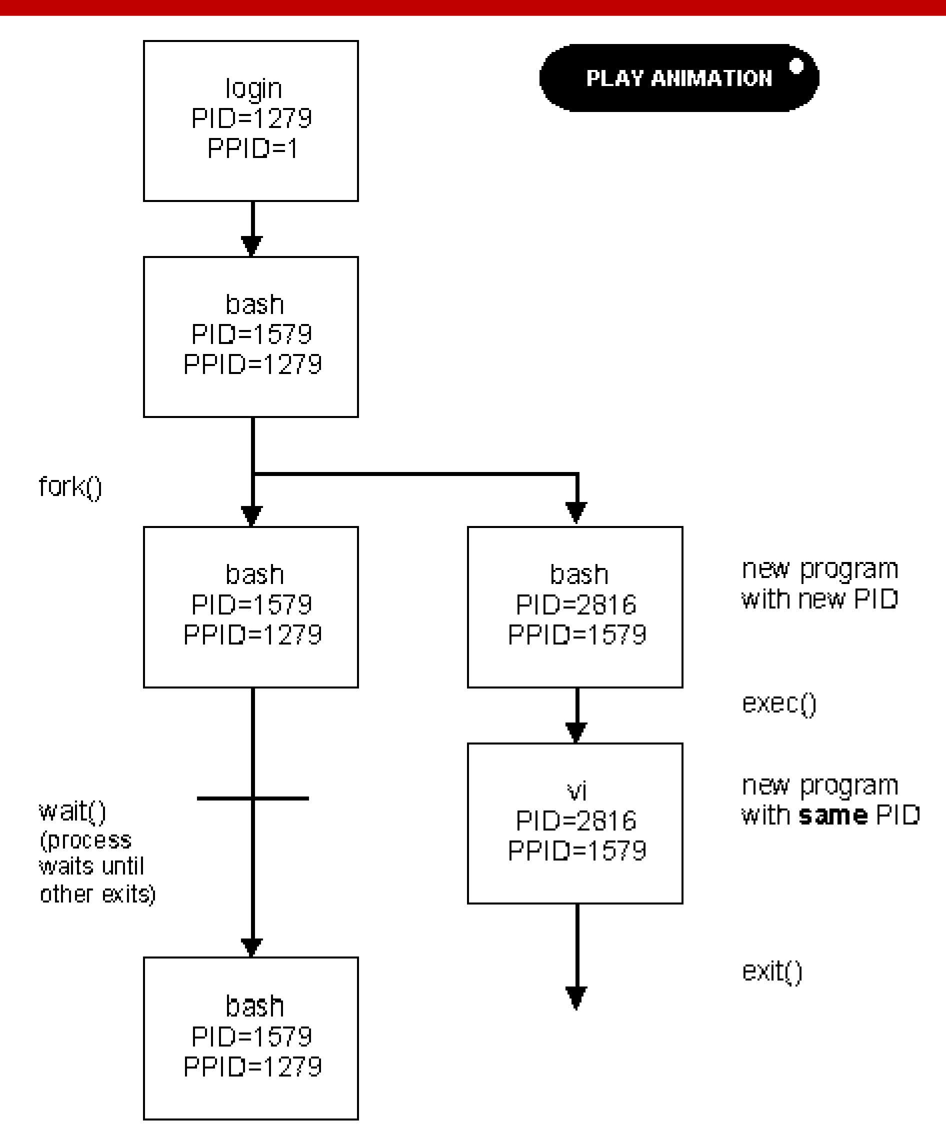
الأن لنفرض أننا بحاجة للقيام بعمليات حسابية طويلة فهذا يؤدي إلى تجمد واجهة البرنامج يمكننا حل هذه المشكلة بمعالجة الرسائل كما في Application.ProcessMessages و لكن يمكننا أيضا الاستفادة من موضوع تعدد الخيوط MultiThreading و في هذه الحالة نقوم بإنشاء مسار فرعي للبرنامج يقوم بالعمليات المطلوبة كما في الصورة التالية



ففي حالتنا هذه يوجد مسار رئيسي للبرنامج هو Main VCL Thread و عند الضغط على الزر يتم توليد مسار فرعي يقوم بالعمليات المطلوبة و هذا المسار الجديد هو عبارة Thread فرعي للبرنامج.

Forking

لكي نفهم ال Forking لنتأمل هذه الصورة



بالبداية ال PID هو عبارة عن ال Process ID أي معرف العملية و هو رقم خاص يدل على عملية معينة فقط أي لكل عملية معرف خاص بها أما ال PPID فهو عبارة عن ال Parent Process ID أي معرف العملية الأب أي الأب الخاص بهذه العملية المتولدة مع ملاحظة أن لكل عملية أب واحد فقط.

الان عندما نقوم بعملية Fork تقوم العملية بتوليد عملية جديدة تسمى ب Child Process أما العملية التي ولدت العملية الجديدة تسمى Parent Process و هكذا نجد أنه تم توليد عملية جديدة بدلا من انشاء خيط كما في Windows و هذه العملية الجديدة اصبح لها معرف خاص و لها أب.

نماذج برمجة الشبكات

هناك نوعان من برمجة ال Sockets تحت بيئة الوندوز هما:

1) Blocking : **الكتلي**

2) Non-Blocking: **غير الكتلي**

و أحيانا تدعى هذه النماذج ب:

1) Blocking: **متزامن** (Synchronous)

2) Non-Blocking: **غير متزامن** (Asynchronous)

تحت بيئة اليونكس النظام المتبع هو Blocking فقط.

بالطبع هناك نماذج أخرى منها I/O Completion Ports **و** Overlapped I/O **و هذه النماذج في العادة تستخدم في برمجة تطبيقات الخوادم المتقدمة. كما أن هذه النماذج ليست مستخدمة في كل الانظمة.**

الاندي 10 يتضمن دعم لهذه النماذج.

الاندي يستخدم هذا النموذج. و استدعاءات هذا النموذج تشبه إلى درجة كبيرة القراءة و الكتابة إلى ملف. فعندما نقوم بقراءة أو كتابة ملف فإن نتيجة الاجراء لا تعود إلا عندما تنتهي العملية.

الفرق بين العمل بالملفات و ال Sockets هو أن العمل بال Sockets ستأخذ وقتا أطول لأن البيانات لن تتوفر مباشرة للقراءة و الكتابة.

في الاندي يكفي استدعاء وحيد لعمل اتصال و ننتظر بعدها نتيجة هذا الاستدعاء ففي حال النجاح فهذا يعني أن الاتصال تم و إلا سيتم استدعاء الاستثناء المناسب.

Non-Blocking Model

يعمل هذا النموذج على نظام من الحوادث. فهنا الاستدعاءات تتم و عندما تكمل هذه الاستدعاءات تكون بحاجة للانتباه لانها تعمل

كمثال:

عندما نريد الاتصال فإنه يتوجب علينا استدعاء منهج الاتصال و هذا الاستدعاء سيعود مباشرة قبل اتمام عملية الاتصال. و عندما تتم عملية الاتصال يتم تفعيل حدث ما. هذا يتطلب أن تكون منطقية الاتصال مجزأة إلى العديد من الاجراءات أو تتم وفق حلقات.

تاريخ Winsock

بالبداية كان هناك بيركيلي يونكس (Berkeley Unix). و كان لديه توابع معيارية Standard Socket API و التي اصبحت متبناة و منتشرة بين توزيعات اليونكس المختلفة.

و بعدها اتى Windows و قرر أحدهم أنه من الجيد أن يكون هناك مقدرة لبرمجة ال TCP/IP في الوندوز. لذلك نقلوا توابع اليونكس الشائعة و هذا مما سمح نقل شيفرات اليونكس الحالية للوندوز بسهولة.

ال Blocking ليست سيئة

يتم بشكل متكرر مهاجمة هذا النموذج استنادا على مقدمات منطقية خاطئة. و على النقيض من الاعتقاد السائد هذا النموذج ليس سيئا.

عندما تم نقل توابع اليونكس للوندوز تم تسمية هذه التوابع ب Winsock (Windows Sockets). و خلال عملية النقل ظهرت المشكلة سريعا.

في اليونكس كان شائعا إنشاء ال Fork (و هو مشابه لتعدد الخيوط Multi -Threading و لكن بعمليات منفصلة بدلا من الخيوط.

عملاء و خوادم اليونكس يقومون بعمل Fork (أي توليد عملية جديدة) لكل اتصال. ثم يتم تنفيذ هذه العمليات بشكل منفصل و يتم استخدام نموذج Blocking.

في نظام Windows 3.1 لم يستطع عمل Fork بشكل مجد و ذلك بسبب نقص دعم تعدد المهام MultiTasking.

كذلك نظام Windows 3.1 لم يكن يدعم الخيوط (Threads). و هذا مما جعل استخدام نموذج Blocking في برمجة التطبيقات أن يسبب في تجمد واجهة المستخدم.

التوجيه Inline

إنطلاقا من دلفي 2005 يمكن أن يضاف التوجيه Inlineعند نهاية تعريف اسم الـ Methode لزيادة سرعة تنفيذها على حساب حجم الذاكرة المحجوزة للـMethode ضمن الملف التنفيذي الناتج.

التوضيح:

أولا يجب أن نعلم أن لكل إصدار دلفي رقم نسخة محدد كما يلي:

```
{$IFDEF VER80}  - Delphi 1
{$IFDEF VER90}  - Delphi 2
{$IFDEF VER100} - Delphi 3
{$IFDEF VER120} - Delphi 4
{$IFDEF VER130} - Delphi 5
{$IFDEF VER140} - Delphi 6
{$IFDEF VER150} - Delphi 7
{$IFDEF VER160} - Delphi 8
{$IFDEF VER170} - Delphi 2005
{$IFDEF VER180} - Delphi 2006
{$IFDEF VER180} - Delphi 2007
{$IFDEF VER185} - Delphi 2007
{$IFDEF VER200} - Delphi 2009
{$IFDEF VER210} - Delphi 2010
{$IFDEF VER220} - Delphi XE
```

إذا كنت تستخدم Delphi و ما بعدها , يمكن كتابة الشرط كما يلي :

```
{$IF CompilerVersion >= 18.5}
//some code only compiled for Delphi 2007 and later
{$IFEND}
Delphi XE   - 22
Delphi 2010 - 21
Delphi 2009 - 20
Delphi 2007 - 18.5
Delphi 2006 - 18
Delphi 2005 - 17
Delphi 8    - 16
Delphi 7    - 15
Delphi 6    - 14
```

نعمل على دلفي 2010 أي 14 نكتب إثنين، إحداهما باستخدام التوجيه Inline

ومن المستحسن أن يكون لهما نفس التعليمات لتسهيل المقارنة فقط.

```
Procedure TestSAS;

Begin

  MessageBoxA(0,'TestSAS',Nil,0);

End ;

Procedure TestSpeedAndSize;Inline;

//{$IF CompilerVersion >= 17}Inline;{$IFEND}

Begin

  MessageBoxA(0,'TestSpeedAndSize',Nil,0);

End ;
```

ننقح كلاهما باستخدام برنامج OllyDbg

```
Procedure TestSAS :
;
Beginning Address was : 004A3BFC
CALL 004A3BE0
RETN
End Address was : 004A3C01;

Beginning Address was : 004A3BE0;
PUSH 0
PUSH 0
PUSH 4A3BF4   // 'TestSAS'
PUSH 0
CALL 0040B994 //MessaageBoxA
RETN
End Address was : 004A3BF0;
```

```
Procedure TestSpeesAndSize :

Beginning Address was : 004A3C04;
PUSH 0
PUSH 0
PUSH 4A3C18   // 'TestSpeedAndSize'
PUSH 0
CALL 0040B994 //MessaageBoxA
RETN
End Address was : 004A3C14;
```

الملاحظات :

الأولى (TestSAS) :

إستدعاء للـ MessageBoxA (**وقت تنفيذ الإستدعاء** T1)

تنفيذ إجراء الـ MessageBoxA (**وقت تنفيذ الإجراء** T2)

وقت التنفيذ المستغرق T = T1 + T2

الثانية (TestSpeedAndSize) :

تنفيذ إجراء الـ MessageBoxA , **مباشرة !**

وقت التنفيذ المستغرق = (**وقت تنفيذ الإجراء** MessageBoxA)

1ـ تنفيذ الدوال التي تحتوي التوجيه *Inline* ***أسرع من تنفيذ الدوال العادية .***

في حالة تكرار إستعمال الدوال التي تحتوي Inline , **سيتكرر الكود في كل مرة تم إستدعائها لذا فإن الحجم النهائي للملف التنفيذي سيزيد ؟, نستدعي** 3 **مرات و نرى!**

```
Beginning Address was : 004A3C58;
PUSH 0
PUSH 0
PUSH 4A3C6C
PUSH 0
CALL 0040B994
RETN
//Som Asm Code
PUSH 0
PUSH 0
PUSH 4A3C94
PUSH 0
CALL 0040B994
RETN
//Som Asm Code
PUSH 0
PUSH 0
PUSH 4A3CBC
PUSH 0
CALL 0040B994
RETN
End Address was : 004A3CB8;
```

كما نلاحظ فإنه تم تكرار الكود في البرنامج , لذا فإن زيادة حجم الملف التنفيذي النهائي حتمية , (لن تلاحظ الفرق في حالة التكرار لمرتين او ثلاث انا أتكلم عن عدد كبير من المرات)

2ـ تنفيذ (Method) الـ Inline يزيد من حجم البرنامج , لتكرار كود التابع عكس الدوال العادية , التي هي عبارة عن إستدعاءات (كود واحد فقط يكفي).

3ـ إذا علينا الموازنة في إستخدام التوجيه Inline , فكما له إجابية فله أيضا سلبية تكرار كود الإستدعاء .

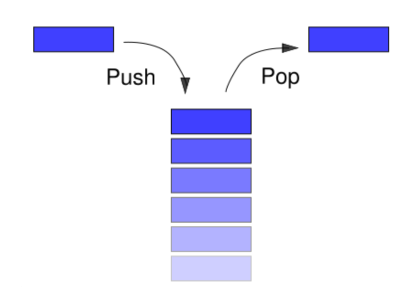
لكي نجعل كود برنامجنا متوافق على حسب نسخ الدلفي , نقوم بإستخدام التوجيه {$IFDEF VER80} **حيث** VER80 **تشير إلى** Delphi 1 .
مثال :

```
function MyFunc(I: Integer): Boolean;{$IFDEF VER210} inline; {$ENDIF}
//Or
Function MyFunc(I: Integer): Boolean;{$IF CompilerVersion >= 17}Inline;
{$IFEND}
```

الأول أخبرنا المترجم أن يقوم بتعريف MyFunc **على أنه** Inline **في حالة كان الإصدار الذي نعمل به** Delphi 2010 , **و الثاني كتابة أخرى تفيد أخذ بالتوجيه** Inline , **في حالة كان إصدار دلفي من** 2005 **و ما تلاه .**

تجسيد للمكدس Stack

تعريف : **المكدس حيز من الذاكرة , على شكل جدول لتصنيف البيانات (إستدعاءات الدوال أو عناوين البارامترات أو قيم ثابتة) , تعمل بطريقة أخر ما يدخل للجدول – أول ما يخرج منه** (LIFO)



تعريف عملي أدق : **سنقوم ببرمجة إجراءات و دوال لمحاكاة التعامل مع الـ** Stack **بنفس مبدأ العمل** , last-in, first-out .

إنشاء مكدس جديد.
إنهاء و تحرير مكدس موجود.
إضافة عنصر للمكدس .
استخراج عنصر من المكدس.
هل المكدس فارغ .
هل المكدس مملوء .
هل أستطيع أن أضيف أو أن أستخرج عنصر من المكدس ؟ .

تجسيد ال Stack **بـجدول** Record :

```
Const
 Stack_Max_Size = 250;
Type
 TStack = Packed Record
   Item : Array [1..Stack_Max_Size] Of String ;
   Stat : Integer;   //Esp
End;
```

إنشاء مكدس جديد : الحالة الابتدائية للمكدس أشرنا إليها بـالقيمة 1-

```
Function NewStack:TStack;
Begin
 Result.Stat:= -1;
End;
```

هل المكدس فارغ :

```
Function StackEmpty(Stack:TStack):Boolean;
Begin
 Result:= (Stack.Stat = -1);
End;
```

هل المكدس مملوء : إذا كان عدد عناصر المكدس يشير الى الحد الأقصى 250

```
Function StackFull(Stack:TStack):Boolean;
Begin
 Result:= (Stack.Stat = Stack_Max_Size);
End;
```

إضافة عنصر للمكدس : تتحقق من أن المكدس غير ممتلئ , في حالة كان اول عنصر سنضيفه و نعيد القيمة الى 1 , و إن لم يكن كذلك نزيد في العداد (Stat) بـ 1 .

```
Procedure AddItem (Stack:TStack;S:String); //Push .
Begin
If Not StackFull(Stack) Then
 Begin
  If Stack.Stat = -1 Then
   Stack.Stat := 1 Else
    Inc(Stack.Stat);
 Stack.Item[Stack.Stat]:= S;
 End;
End;
```

إستخراج عنصر من المكدس : تتحقق أولا من ان المكدس ليس خال , ومن ثم نعمل إفراغ للخانة الأخيرة

```
Procedure ExtrItem (Stack:TStack); //Pop.
Begin
If Not StackEmpty(Stack) Then
 Begin
  Stack.Item[Stack.Stat]:= '';
  If Stack.Stat > 1 Then
   Dec(Stack.Stat)Else
    Stack.Stat := -1;
 End;
End;
```

إنهاء و تحرير مكدس موجود :

```
Procedure FreeStack (Stack:TStack);
Begin
While Not StackEmpty(Stack) Do
 ExtrItem (Stack);
Stack.Stat:= -1;
End;
```

ما هي أخر قيمة مدخلة للمكدس :

```
Function LastInItem (Stack:TStack): String;
Begin
If StackEmpty(Stack) Then
 Result:= '' Else
  Result:= Stack.Item[Stack.Stat];
End;
```

حفظ المكدس في ملف نصي أو في TStrings :

```
Procedure SaveStack (Stack:TStack;DisTab:TStrings);
Var
 _i: Byte;//<=250.
Begin
DisTab.Clear;
If Stack.Stat = -1 Then
 DisTab.Add('[Stack Empty]')
Else
 Begin
  If Stack.Stat = Stack_Max_Size Then
   DisTab.Add('[Stack Full]')
  Else
   For _i:= Stack.Stat DownTo 1 Do
    DisTab.Add(Stack.Item[_i]);
 End;
End;
```

الخلاصة :

قمنا بمحاكاة لعمل المكدس , لم نتطرق فيها الى التفاصيل، أيضا إستخدمنا فقط عناصر من نمط واحد (String) لتسهيل التطبيق لفهم أكثر لعملية الـ Push/Pop.

قواعد البيانات – بقلم كارلوس هـ. كانتو ترجمة خالد الشقروني

تعرف على فايربيرد Firebird بدقيقتين



مقدمة

إذا كنت تقرأ هذه المقالة، فمن المحتمل أنها المرة الأولى التي تتعرّف فيها على فايربيرد نظام إدارة قواعد البيانات العلائقية. ستعرض لك هذه الورقة الميزات الرئيسية لقواعد بيانات فايربيرد. وأنا متأكد أنك في النهاية ستكون توّاقا لتنزيل مثبتها الخفيف لتجربها بنفسك.

نبذة تاريخية

انبثقت فايربيرد من الشفرة المصدرية لقاعدة بيانات انتربيس 6.0 من بورلاند. وهي مفتوحة المصدر ولا يوجد بها ترخيص مزدوج. وسواء احتجت إليها لتطبيقات تجارية أو مفتوحة المصدر، فهي مجانا بالكامل!

تقنية فايربيرد قيد الاستخدام منذ 20 عاما، مما يجعلها منتجا شديد النضوج والاستقرار

المميزات الرئيسية

لا يغرنك حجم المثبت! فايربيرد هو نظام قوي ومكتمل المزايا لإدارة قواعد البيانات العلائقية. ويمكنه مناولة قواعد بيانات حجمها من مجموعة كيلو بايت فقط إلى العديد من الغيغابايت مع أداء جيد ودون الحاجة إلى صيانتها في أغلب الأحوال!

فيما يلي قائمة ببعض أهم مزايا فايربيرد:

- **دعم كامل للإجراءات المخزونة** Stored Procedure **والمفعلات** Triggers
- **تلبية كاملة لعمليات** ACID (**الوحدانية، التجانس، العزل، المتانة**)
- **تكامل مرجعي** Referential Integrity
- **معمارية متعددة الأجيال** Multi Generational Architecture
- **بصمة صغيرة جدا**
- **لغة داخلية مكتملة المزايا للإجراءات المخزونة والمفعلات PSQL.**
- **دعم الإجرائيات الخارجية** UDF.
- **ندرة الحاجة لوجود مدراء قواعد بيانات متخصصين DBAs**
- **لا حاجة للتوصيفات تقريبا – فقط قم بتثبيته وابدأ التشغيل!**
- **مجتمع كبير لفايربيرد والعديد من الأماكن التي تجد فيها دعما مجانيا جيدا.**
- **خيار استخدام نسخة مدمجة** embedded **بملف وحيد – خيار جيد لإنشاء تطبيقات على القرص المدمج، أو لمستخدم واحد، أو تلك الخاصة بالعرض والتقييم.**
- **العشرات من الأدوات من مصادر خارجية تتضمنها أدوات رسومية للتحكم والإدارة، وأدوات توأمة البيانات، ألخ.**
- **الكتابة بعناية – استعادة سريعة، دون الحاجة لملفات تدوين العمليات** logs !
- **عدة طرق للنفاذ إلى قاعدة البيانات تتضمن الدوال الأصلية** native/API **أو مسيرات** dbExpress **أو مزودات** ODBC **و** OLEDB **و دوت نت و مسيرات** JDBC **وقوالب لبايثون** Python **و** PHP **و** Perl **، الخ.**
- **دعم جذري لأنظمة التشغيل الرئيسية بما فيها و يندوز و لينوكس و سولاريس و ماك او اس و** HP-UX **و** FreeBSD.
- **نسخ احتياطي مركب تزايدي** Incremental Backups
- **متوفر ببنية 64 بت**
- **تنفيذ كلي للمؤشر** cursor **في لغة** PSQL
- **جداول مراقبة**
- **مفعلات** Trigers **عند الاتصال و انجاز العمليات** Transactions
- **جداول مؤقتة**
- **دوال التتبع** TraceAPI **– لمعرفة ماذا يحدث في خادم قاعدة البيانات**

جربها الآن!

تجربة فايربيرد مهمة سهلة جدا. يصل حجم المثبت إلى أقل من 7 م ب (حسب نظام التشغيل) كما أنه آلي بالكامل. يمكنك تنزيله من موقع فايربيرد الرئيسي.

ستلاحظ أن خادم فايربيرد يأتي بأربعة نكهات: خادم ممتاز SuperServer، تقليدي Classic، تقليدي ممتاز SuprerClassic و مدمج Embeded. يمكنك البدء بالخادم الممتاز الآن، ينصح بالتقليدي لأغراض الاستخدام مع أجهزة SMP وبعض الحالات الخاصة الأخرى. يتقاسم الخادم الممتاز مخزونه cache مع حالات الاتصال بقاعدة البيانات ويستخدم المسارات/الخيوط threads للتعامل مع كل اتصال. ويقوم التقليدي ببدء عملية معالجة خادم مستقلة لكل اتصال قائم. بينما يوفر التقليدي الممتاز معالجة خادم واحدة ومتعددة المسارات مع مخزون مستقل لكل اتصال.

النسخة المدمجة embedded هي تنوع مذهل للخادم. فهي خادم فايربيرد مكتمل المزايا محزمة في ملفات معدودة. وهي بذلك سهلة عند نشرها , حيث لا توجد حاجة لتركيب وتثبيت الخادم. هي تلائم عروض الأقراص المدمجة، ونسخ العرض والتقييم للبرامج والتطبيقات المكتبية القائمة بذاتها.

تأتي فايربيرد مع مجموعة كاملة من أدوات الأوامر النصية التي تسمح لك بإنشاء قواعد بيانات واستخلاص إحصائيات عنها، وتنفيذ أوامر SQL أو تعليمات نصية scripts ، كذلك إجراء عمليات النسخ الاحتياطي والاسترجاع، إلى آخر ذلك من العمليات. وإذا كنت تفضل استخدام أدوات بواجهة استخدام رسومية ، فهناك العديد من الخيارات يمكنك الاختيار من بينها بما فيها البرامج المجانية. راجع القائمة في آخر هذه الورقة.

في ويندوز يمكنك تشغيل فايربيرد كخدمة أو كتطبيق المثبت يمكنه إنشاء أيقونة في لوحة التحكم Control Panel لاستخدامها لإدارة الخادم (ابتداء، إيقاف، الخ..).

التوثيق

يوجد العديد من الأوراق، و إجابات الأسئلة المتكررة FAQs والمقالات التي قد ترغب بإلقاء نظرة عليها في موقع فايربيرد الرئيسي. أيضا، يمكنك تفقد احتمال وجود مواقع أو منتديات بلغتك، من أجل الحصول على المساندة و الدعم.

كل هذه المعلومات يمكن ايجادها هنا أو هناك في موقع فايربيرد الرئيسي. أيضا راجع الموقع www.firebirdnews.org للحصول آخر مستجدات الأخبار المتعلقة بفايربيرد.

لجميع أحجام قواعد البيانات.

البعض يعتقد أن فايربيرد هو نظام إدارة قواعد بيانات علائقية تستخدم فقط مع قواعد البيانات صغيرة الحجم بخطوط اتصال محدودة. أنهم مخطئون لقد تم استخدام فايربيرد مع العديد من قواعد البيانات الضخمة و بالكثير من خطوط الاتصال. يمكنك قراءة مقالة كاملة حول قاعدة بيانات حقيقية بحجم 1 تيرابايت.